Regstar: Efficient Strategy Synthesis for Adversarial Patrolling Games

David Klaška[1] **Antonín Kučera**[1] **Vít Musil**[1] **Vojtěch Řehák**[1]

[1]Faculty of Informatics, Masaryk University, Brno, Czech Republic

Abstract

We design a new efficient strategy synthesis method applicable to adversarial patrolling problems on graphs with arbitrary-length edges and possibly imperfect intrusion detection. The core ingredient is an efficient algorithm for computing the value and the gradient of a function assigning to every strategy its "protection" achieved. This allows for designing an efficient strategy improvement algorithm by differentiable programming and optimization techniques. Our method is the first one applicable to real-world patrolling graphs of reasonable sizes. It outperforms the state-of-the-art strategy synthesis algorithm by a margin.

1 INTRODUCTION

Patrolling games are a special type of security games where a mobile Defender moves among vulnerable targets and aims to detect possible ongoing intrusions initiated by an Attacker. The targets are modelled as vertices in a directed graph where the edges correspond to admissible Defender's moves. At any moment, the Attacker may choose some target τ and initiate an intrusion (attack) at τ. Completing this intrusion takes $d(\tau)$ time units, and if the Defender does not visit τ in time, he is penalized by utility loss determined by the cost of τ.

In *adversarial* patrolling games [Vorobeychik et al., 2012, Agmon et al., 2008a, 2009, Basilico et al., 2012, 2009, de Cote et al., 2013, Lin et al., 2019], the Attacker knows the Defender's strategy and can even observe the Defender's moves[1]. These assumptions are particularly appropriate in situations where the actual Attacker's abilities are *unknown* and the Defender is obliged to guarantee a certain level of protection even in the worst case. This naturally leads to using *Stackelberg equilibrium* [Sinha et al., 2018, Yin et al., 2010] as the underlying solution concept, where the Defender/Attacker play the roles of the leader/follower, i.e., the Defender commits to a moving strategy γ, and the Attacker follows by selecting an appropriate counter-strategy π. The *value* of γ, denoted by $\mathrm{Val}(\gamma)$, is the expected Defender's utility guaranteed by γ against an arbitrary Attacker's strategy. Intuitively, $\mathrm{Val}(\gamma)$ corresponds to the "level of protection" achieved by γ.

The basic algorithmic problem in patrolling games is to compute a Defender's strategy γ such that $\mathrm{Val}(\gamma)$ is as large as possible. Since general history-dependent strategies are not algorithmically workable (see Sec. 3.1), recent works [Kučera and Lamser, 2016, Klaška et al., 2018] concentrate on computing *regular* strategies where the Defender's decisions depend on finite information about the history of previously visited vertices. As Kučera and Lamser [2016] observed, regular strategies provide better protection than *memoryless* strategies where the Defender's decision depends only on the currently visited vertex. However, the mentioned algorithms apply only to patrolling graphs where all edges have the *same* length (traversal time). A longer distance between vertices can be modelled only by adding a sequence of auxiliary vertices and edges, quickly pushing

Figure 1: We synthesize an efficient strategy for patrolling the net of Montreal's ATMS.

[1]The Defender may choose the next move randomly according to a distribution specified by its moving strategy. Although the Attacker knows the Defender's strategy (i.e., the distribution), it *cannot* predict the way of resolving the randomized choice.

Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence (UAI 2021), PMLR 161:471–481.

the resulting graph's size beyond the edge of feasibility (see Sec. 5). Even the currently best algorithm of Klaška et al. [2018] fails to solve small real-world patrolling graphs such as a network of selected ATMs in Montreal (Fig. 1).

Our contribution

1. We prove that **regular Defender's strategies** are not only better than memoryless strategies, but they are **arbitrarily close** to the power of general strategies. Therefore, the scope of strategy synthesis can be safely restricted to regular strategies. This resolves the open question of previous works.
2. We design an efficient REGSTAR[2] algorithm for computing regular Defender's strategies in general patrolling graphs with edges of **arbitrary length**. REGSTAR applies to scenarios with **imperfect intrusion detection**, where the probability of discovering an intrusion at a target τ by the Defender visiting τ is not necessarily equal to one.
3. We validate REGSTAR experimentally. We compare REGSTAR against the best existing algorithm when applicable, and we perform a set of **real-life experiments** demonstrating its strengths and limits.

We depart from the fact that the function Val assigning the protection value to a given regular Defender's strategy is *differentiable*. The very heart of our algorithm is a novel, efficient procedure for computing the value and the gradient of Val. Since the size of the closed-form expression representing Val is exponential, the task is highly non-trivial. We apply differentiable programming techniques and design an efficient strategy improvement algorithm for regular strategies based on gradient ascent. The REGSTAR algorithm randomly generates many initial regular strategies, improves them, and returns the best strategy.

The efficiency of REGSTAR is evaluated experimentally in Sec. 5. In the first series of experiments, we compare the efficiency of REGSTAR against the algorithm of Klaška et al. [2018]. In the second series, we demonstrate the applicability of REGSTAR to a real-world patrolling graph with 18 targets corresponding to selected ATMs in Montreal. In the last series, we document the power of regular strategies on patrolling graphs modelling buildings with corridors and offices. Here, the information about the history of visited vertices is crucial for achieving reasonable protection.

Experiments prove that REGSTAR outperforms the method of Klaška et al. [2018] and can solve instances far beyond the reach of this algorithm. Our approach adopts the *infinite horizon* patrolling game model, and therefore it does not suffer from the scalability issues caused by increasing the time bound in finite-horizon security games (see Sec. 2 for more comments). For practical applications, solving patrolling graphs with about 20–30 targets seems sufficient, as the protection achievable by a single Defender becomes low for a higher number of targets, and the patrolling task needs to be solved by multiple Defenders[3].

[2]REGular STrategy ARchitect.

2 RELATED WORK

Patrolling games are a special type of *security games* where game-theoretic concepts are used to determine the optimal use of limited security resources [Tambe, 2011]. Security games with static allocation have been studied in, e.g., [Jain et al., 2010, Kiekintveld et al., 2009, Pita et al., 2008, Tsai et al., 2009, Xu et al., 2018, 2015, Gan et al., 2017]. For patrolling games, where the Defender is mobile, most of the existing works assume the Defender is following a *positional* strategy that depends solely on the current position of the Defender [Basilico et al., 2012]. Since positional strategies are weaker than general history-dependent strategies, there were also attempts to utilize the history of the Defender's moves. This includes the technique of duplicating each node of the graph to distinguish internal states of the Defender (for example, Agmon et al. [2008a] consider a direction of the patrolling robot as a specific state; this is further generalized in Bošanský et al. [2012]). Another concept is higher-order strategies [Basilico et al., 2009], where the Defender takes into account a bounded sequence of previously visited states. In our work, we use regular strategies [Kučera and Lamser, 2016, Klaška et al., 2018], where the information about the history of Defender's moves is abstracted into finitely many memory elements.

The existing strategy synthesis algorithms for patrolling games are based either on (1) mathematical programming with non-linear constraints, or (2) restricting the graph topology to some manageable subclass, or (3) strategy improvement, or (4) reinforcement learning. The first approach (see, e.g., [Basilico et al., 2012, 2009, Bošanský et al., 2011]) suffers from scalability issues. In Bošanský et al. [2011], the authors consider mobile targets, which forces the strategy to be time-dependent. Basilico et al. [2012] consider higher-order strategies in theory, but they perform experiments with positional strategies only due to computational infeasibility (one can easily construct examples where positional strategies are weaker than regular strategies and the protection gap is up to 100%. The experiments of Vorobeychik et al. [2012] are also limited to positional strategies. Lin et al. [2019] make full use of the history, yet they study only perimeters (i.e., cycles), so their approach does not apply to graphs with arbitrary topologies.

The second approach applies only to selected topologies, such as lines, circles [Agmon et al., 2008a,b], or fully connected graphs with unit distance among all vertices [Brázdil

[3]Patrolling by multiple Defenders is studied independently, see, e.g., [Beynier, 2017, Gan et al., 2018]

et al., 2018]. The third approach has been applied only to patrolling graphs with edges of unit length. The algorithm of Kučera and Lamser [2016] requires a certain level of human assistance because the underlying finite-state automaton gathering the information about the Defender's history must be handcrafted. Klaška et al. [2018] overcome this limitation by designing an automatic strategy synthesis algorithm, and it is the most efficient strategy synthesis procedure existing before our work. There is a high-level similarity to our algorithm because both use a variant of gradient ascent and construct regular strategies. However, the internals of the two algorithms is different. The core of our method is a novel procedure for computing the value and the gradient of the value function. In particular, our algorithm avoids the blowup in the number of states when modeling real-world patrolling scenarios with variable length edges. This allows for processing instances far beyond the reach of the algorithm of Klaška et al. [2018], as documented experimentally in Sec. 5.2.

The fourth approach has been successful mainly for games with the finite horizon [Wang et al., 2019, Karwowski et al., 2019] and suffers from the exponential blowup in the number of finite paths with the increasing time bound.

3 PATROLLING GAMES

We recall the standard model of adversarial patrolling games and fix the notation used.

Patrolling graphs A *patrolling graph* is a tuple $G = (V, T, E, time, d, \alpha, \beta)$, where

- V is a non-empty set of *vertices*;
- $T \subseteq V$ is a non-empty set of *targets*;
- $E \subseteq V \times V$ is a set of *edges*;
- $time\colon E \to \mathbb{N}$ denotes the time to travel an edge;
- $d\colon T \to \mathbb{N}$ specifies the time to complete an attack;
- $\alpha\colon T \to \mathbb{R}_+$ defines the costs of targets;
- $\beta\colon T \to (0, 1]$ is the probability of a successful intrusion detection.

For short, we write $u \to v$ instead of $(u, v) \in E$, and denote $\alpha_{\max} = \max_{\tau \in T} \alpha(\tau)$ and $d_{\max} = \max_{t \in T} d(t)$. In the sequel, let G be a fixed patrolling graph.

3.1 DEFENDER'S STRATEGY

A Defender's strategy is a recipe for selecting the next vertex. In general, the Defender may choose the next vertex randomly depending on some information about the history of previously visited vertices.

Let $\mathcal{H}$ be the set of all finite paths in G, including the empty path λ. A *Defender's strategy* for G is a function $\gamma\colon \mathcal{H} \to Dist(V)$ where $Dist(V)$ is the set of all probability distributions on V such that whenever $\gamma(h)(v) > 0$, then either $h = \lambda$ or $u \to v$ where u is the last vertex of h. Note that $\gamma(\lambda)$ corresponds to the initial distribution on V.

Unrestricted Defender's strategies may depend on the whole history of previously visited vertices when selecting the next vertex, and they may not be finitely representable.

3.2 ATTACKER'S STRATEGY

We consider the same patrolling game as in Klaška et al. [2020], where the time is spent by moving along the edges. We also use the same notion of Attacker's strategy, assuming that, in the *worst case*, the Attacker can determine the next edge taken by the Defender *immediately* after the Defender leaves the currently visited vertex. This means that the Attacker's decision is based not only on the history of visited vertices but also on edge taken next.

The Attacker cannot gain anything by delaying his attack until the Defender arrives at the next vertex. Therefore we can assume an attack is initiated at the moment when the Defender leaves the currently visited vertex. Furthermore, the Attacker can attack *at most once* during a play[4].

An *observation* is a finite sequence $o = v_1, \ldots, v_n$, $v_n \to v_{n+1}$, where $v_1, \ldots, v_n \in \mathcal{H}$. The set of all observations is denoted by Ω. An *Attacker's strategy* for a patrolling graph G is a function $\pi\colon \Omega \to \{wait, attack_\tau : \tau \in T\}$. We require that if $\pi(v_1, \ldots, v_n, v_n \to u) = attack_\tau$ for some $\tau \in T$, then $\pi(v_1, \ldots, v_i, v_i \to v_{i+1}) = wait$ for all $1 \le i < n$ ensuring that the Attacker can attack at most once.

3.3 EVALUATING DEFENDER'S STRATEGY

Let $o = v_1, \ldots, v_n$, $v_n \to v_{n+1}$, $n \ge 1$, be an observation, $\tau \in T$ a target and consider an attack at τ after observing o.

By $Path(o, \tau)$ we denote the set of all finite paths u from v_{n+1} to $v_{n+1+k} = \tau$ of the length $k \ge 0$ such that the total time needed to traverse from v_n to τ along u does not exceed $d(\tau)$.

For $u \in Path(o, \tau)$, let $\mathrm{Eval}(\tau \mid u)$ be the value defended at τ when the Defender discovered the attack in the last vertex of u. The probability of detecting the attack at τ after traversing u equals $(1 - \beta(\tau))^{\#_\tau(u)-1} \cdot \beta(\tau)$, where $\#_\tau(u)$ stands for the number of visits to τ along u. Indeed, since

[4] Even if the Attacker can perform another attack after completing the previous one, the best choice the Defender is to follow an optimal strategy constructed for the single attack scenario. This is no longer true when the Defender has to spend some time responding to the discovered attack [Lin et al., 2019] or when multiple Attackers can perform several attacks concurrently.

the intrusion detection is not perfect, the Defender failed to discover the attack at the first $\#_\tau(u) - 1$ trials with the probability $1 - \beta(\tau)$ and succeeded at the last one with the probability $\beta(\tau)$. Specially, when $\beta = 1$ and $\#_\tau(u) = 1$, the factor becomes 0^0, interpreted as 1. Therefore, as $\alpha(\tau)$ denotes the cost of τ,

$$\mathrm{Eval}(\tau \mid u) = \alpha(\tau) \cdot (1 - \beta(\tau))^{\#_\tau(u)-1} \cdot \beta(\tau). \quad (1)$$

The *protection* achieved by γ against an attack at τ initiated after observing o is defined as

$$\mathbf{P}^\gamma(\tau \mid o) = \sum_{u \in Path(o,\tau)} \mathrm{Prob}^\gamma(u \mid o) \cdot \mathrm{Eval}(\tau \mid u), \quad (2)$$

where $\mathrm{Prob}^\gamma(u \mid o)$ is the probability of performing u after observing o, i.e.,

$$\mathrm{Prob}^\gamma(u \mid o) = \prod_{i=1}^{k} \gamma(v_1, \ldots, v_{n+i})(v_{n+i+1}). \quad (3)$$

Similarly, we use $\mathrm{Prob}^\gamma(o)$ to denote the probability that observation o occurs, i.e.,

$$\mathrm{Prob}^\gamma(o) = \prod_{i=0}^{n} \gamma(v_1, \ldots, v_i)(v_{i+1}). \quad (4)$$

Now let π be an Attacker's strategy. For every $\tau \in T$, let $Att(\pi, \tau)$ be the set of all $o \in \Omega$ such that $\pi(o) = attack_\tau$. The *expected Attacker's utility* for γ and π is defined as

$$\mathbb{E}U_A(\gamma, \pi) = \sum_{\tau \in T} \sum_{o \in Att(\pi,\tau)} \mathrm{Prob}^\gamma(o) \cdot \big[\alpha(\tau) - \mathbf{P}^\gamma(\tau \mid o)\big]. \quad (5)$$

Note that $\mathbb{E}U_A(\gamma, \pi)$ corresponds to the expected amount "stolen" by the Attacker. Consistently with Klaška et al. [2018], the *expected Defender's utility* is defined as

$$\mathbb{E}U_D(\gamma, \pi) = \alpha_{\max} - \mathbb{E}U_A(\gamma, \pi)\,. \quad (6)$$

The Defender/Attacker aims to maximize/minimize the expected protection $\mathbb{E}U_D(\gamma, \pi)$, respectively. The *value* of a given Defender's strategy γ is the expected protection guaranteed by γ against an *arbitrary* Attacker's strategy, i.e.,

$$\mathrm{Val}_G(\gamma) = \inf_\pi \mathbb{E}U_D(\gamma, \pi)\,. \quad (7)$$

Maximal protection achievable in G is then

$$\mathrm{Val}_G = \sup_\gamma \mathrm{Val}_G(\gamma)\,. \quad (8)$$

4 THE METHOD

Our approach consists of three stages. First, since the value function from (7) is not a priori tractable, we analyze the value of *regular strategies* for G. This yields a closed-form differentiable value function. Secondly, we design an *efficient algorithm* that computes the value function and its gradient. This is the heart of our contribution. Finally, we combine these elements into the RegStar algorithm, which *generates the best regular strategies* via gradient ascent.

4.1 REGULAR DEFENDER'S STRATEGIES

An algorithmically workable subclass of Defender's strategies are *regular* strategies used by Kučera and Lamser [2016], Klaška et al. [2018]. In this concept, the information about the history of visited vertices is abstracted into a *finite* set of memory elements assigned to each vertex.

Formally, we turn vertices $v \in V$ of G into *eligible pairs* $\widehat{V} = \{(v, m) : v \in V,\ 1 \le m \le \mathrm{mem}(v)\}$ in which the memory sizes $\mathrm{mem}\colon V \to \mathbb{N}$ are fixed. A *regular Defender's strategy* for G is a function $\sigma\colon \widehat{V} \to Dist(\widehat{V})$ satisfying $\sigma(v, m)(v', m') > 0$ only if $v \to v'$. Intuitively, the Defender traverses the vertices of G updating memory elements and thus "gathering" some information about the history of visited vertices. The probability that v' with information represented by m' is visited after the current v with information m is given by $\sigma(v, m)(v', m')$. A regular strategy σ is called *deterministic-update* if $\sigma(v, m)(v', m_1) > 0$ and $\sigma(v, m)(v', m_2) > 0$ imply $m_1 = m_2$. This means that when the Defender is in (v, m), he may randomize to choose the next vertex v', but the next memory element is then determined uniquely by (v, m) and v'.

For a regular strategy σ, we derive an expression $\mathrm{RVal}_G(\sigma)$ which corresponds to the value of σ against an Attacker who can observe even the current memory element. This approach is consistent with the worst-case paradigm discussed in Sec. 1, as it is not clear whether the Attacker is capable of that. We show that the expression $\mathrm{RVal}_G(\sigma)$ is indeed a lower bound on $\mathrm{Val}_G(\sigma)$ (i.e., σ's value against an Attacker who *cannot* observe the current memory element, c.f. (7)) and under reasonable assumptions, they are equal (Claim 1). Since $\mathrm{RVal}_G(\sigma)$ is in a closed form, this makes regular strategies algorithmically workable. Furthermore, we show that regular strategies can achieve protection *arbitrarily close* to the *optimal* protection achievable by unrestricted strategies. Thus, they offer a convenient trade-off between optimality and tractability.

Theorem 1. *Let G be a patrolling graph and Reg the class of all regular Defender's strategies in G. Then*

$$\sup_{\sigma \in Reg} \mathrm{Val}_G(\sigma) = \mathrm{Val}_G \quad (9)$$

Proof (sketch). First, we fix an *optimal* Defender's strategy $\gamma : \mathcal{H} \to Dist(V)$ satisfying $\mathrm{Val}_G(\gamma) = \mathrm{Val}_G$ (the existence of γ has been proven in [Brázdil et al., 2015]).

To every history h, we associate a finite tree T_h of depth $d_{\max}$ such that:

- The set of nodes contains all histories h' such that hh' is also a history and the length of h' is bounded by $d_{\max}$;
- The root of T_h is the empty history λ;

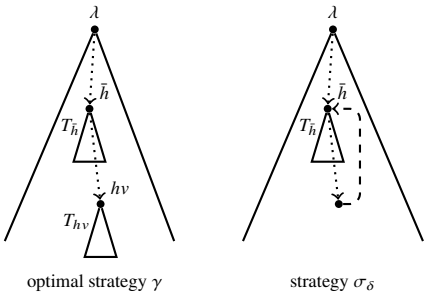

Figure 2: Folding γ into σ_δ.

- $h' \xrightarrow{x} h'v$ is an edge of T_h iff the distribution $\gamma(hh')$ selects the vertex v with probability $x > 0$.

We say that histories $h, \bar{h}$ are δ-*similar* for a given $\delta > 0$ iff $h, \bar{h}$ end in the same vertex and the trees T_h and $T_{\bar{h}}$ are the same up to δ-bounded differences in edge probabilities. Observe that one can construct a fixed sequence of finite trees $T_1, \ldots, T_n$ such that *every* T_h is δ-similar to some T_i, where the n depends just on G and δ. Furthermore, we say that a history h *has* δ-*similar prefix* $\bar{h}$ iff $h = \bar{h}w$ where $h, \bar{h}$ are δ-similar and the length of w is larger than $d_{\max}$.

Let H_δ be the set of all histories h such that $\mathrm{Prob}^\gamma(h) > 0$ and no prefix of h (including h itself) has a δ-similar prefix. Observe that the maximal length of such a history is bounded by $n \cdot (d_{\max}{+}1)$ where the n is defined above, and hence H_δ is a finite set. Let mem be a function assigning $|H_\delta|$ memory elements to every vertex. To simplify our notation, we identify memory elements with the elements of H_δ. Now consider the regular strategy σ_δ defined as follows: for every eligible pair (v, h), we have that $\sigma_\delta(v, h)(v', h') = x$ iff $\gamma(\widehat{h})(v') = x$ and $h' = \widehat{h}$, where

$$\widehat{h} = \begin{cases} \bar{h} & \text{if } hv \text{ has a } \delta\text{-similar prefix } \bar{h}, \\ hv & \text{otherwise.} \end{cases}$$

Intuitively, σ_δ is obtained by "folding" the optimal strategy γ after encountering a history hv with a δ-similar prefix $\bar{h}$, see Fig. 2.

The proof is completed by showing that $\lim_{\delta \to 0^+} \mathrm{Val}_G(\sigma_\delta) = \mathrm{Val}_G$. This is intuitively plausible, because σ_δ becomes more similar to γ for smaller δ. However, the argument also depends on the subgame-perfect property of optimal strategies.

Since σ_δ is a deterministic-update regular strategy, Theorem 1 holds even for the subclass of deterministic-update regular strategies. □

Let us fix a regular strategy σ. For convenience, we write $\sigma(\widehat{u}, \widehat{v})$ for $\sigma(\widehat{u})(\widehat{v})$. The set of *eligible edges* $\widehat{E}$ consists of all edges $e \in \widehat{V} \times \widehat{V}$ such that $\sigma(e) > 0$. These are exactly the edges actually used by the Defender. A finite sequence of eligible pairs $u = (v_1, m_1), \ldots, (v_n, m_n)$ is called an *eligible path* if $v_1, \ldots, v_n$ is a path in G. The probability of executing u is

$$\mathrm{Prob}^\sigma(u) = \prod_{i=1}^{n-1} \sigma\big((v_i, m_i), (v_{i+1}, m_{i+1})\big) \tag{10}$$

For $e = ((v, m), (v_1, m_1)) \in \widehat{E}$ and $\tau \in T$, let $\mathit{Path}(e, \tau)$ denote the set of all eligible paths $(v_1, m_1), \ldots, (v_k, m_k)$ such that $v_k = \tau$, $k \geq 1$ and the traversal time of the path $v, v_1, \ldots, v_k$ is at most $d(\tau)$.

We can now express the value of a regular strategy σ as follows:

$$\mathrm{RVal}_G(\sigma) = \alpha_{\max} - \max_{e \in \widehat{E},\, \tau \in T} \big\{\alpha(\tau) - \mathbf{P}^\sigma(e, \tau)\big\}, \tag{11}$$

where

$$\mathbf{P}^\sigma(e, \tau) = \sum_{u \in \mathit{Path}(e, \tau)} \mathrm{Prob}^\sigma(u) \cdot \mathrm{Eval}(\tau \mid u). \tag{12}$$

Claim 1. *Let σ be a regular strategy. Then*

$$\mathrm{Val}_G(\sigma) \geq \mathrm{RVal}_G(\sigma) \tag{13}$$

Moreover, if the graph $(\widehat{V}, \widehat{E})$ is strongly connected and σ is deterministic-update, then the above holds with equality.

Proof (sketch). It is easy to observe that the edge lengths, the target cost $\alpha(\tau)$ and the detection probability $\beta(\tau)$ are correctly accounted for in the definition of $\mathbf{P}^\sigma(e, \tau)$. The rest of the argument is the same as in [Kučera and Lamser, 2016, Klaška et al., 2018]. □

A regular strategy σ depends only on reasonably many variables $\sigma(e)$, $e \in \widehat{E}$. Hence we identify σ as an element of $\mathbb{R}^{|\widehat{E}|}$. The function $\mathrm{RVal}_G : \mathbb{R}^{|\widehat{E}|} \to \mathbb{R}$ of variable σ is differentiable up to the set where the points of maxima in (13) are not unique.

Claim 1 equips us with a closed-form formula for strategy evaluation. Hence, it enables us to apply methods from differentiable programming to obtain the value of a strategy and its sensitivity to the change of the input.

4.2 STRATEGY EVALUATION

We describe our algorithm that evaluates RVal_G and its gradient at a given point σ. According to (13), we first evaluate all the protection values $\mathbf{P}^\sigma(e, \tau)$ and their gradients $\nabla \mathbf{P}^\sigma(e, \tau)$. The value $\mathrm{RVal}_G(\sigma)$ and its gradient should then be simply the smallest of all values and its gradient. In practice, we replace this minimum with its "soft" version. The details are addressed in Sec. 4.3.

Computing $\mathbf{P}^\sigma$ and $\nabla\mathbf{P}^\sigma$ Given $e \in \widehat{E}$ and $\tau \in T$, a naive approach based on *explicitly constructing* $\mathbf{P}^\sigma(e,\tau)$ is inevitably inefficient, because the set $Path(e,\tau)$ may contain exponentially many different paths. We overcome this problem by performing a search on the graph during which (sub)paths with the same endpoints and the same traversal time are "aggregated" into a single term, resulting in a more compact representation of $\mathbf{P}^\sigma(e,\tau)$ and $\nabla\mathbf{P}^\sigma(e,\tau)$. The search is guided by a min-heap $\mathcal{H}$, similarly as in Dijkstra's shortest path algorithm. However, unlike in Dijkstra's, where the search is initiated from each vertex at most once, we must consider *all* paths from e to τ whose traversal time does not exceed $d(\tau)$, and we must keep track of their probability and the corresponding gradient. To that purpose, each item of $\mathcal{H}$ corresponds to a certain *set* of paths. Moreover, instead of initiating the search at e, we initiate it at τ and search the graph backwards. This trick allows us to compute $\mathbf{P}^\sigma(e,\tau)$ for a given $\tau \in T$ and *all* $e \in \widehat{E}$ at once, thereby saving a factor of $|\widehat{E}|$ in the resulting time complexity.

In particular, for any $(v,m) \in \widehat{V}$ and $t \le d(\tau)$, let $\mathcal{L}_{v,m,t}$ denote the set of all paths which start at (v,m), end at $(\tau,\cdot)$ and have traversal time t. Further, for any $\mathcal{L} \subseteq \mathcal{L}_{v,m,t}$, let

$$\mathbf{P}^\sigma(\mathcal{L}) = \sum_{u \in \mathcal{L}} \mathrm{Prob}^\sigma(u) \cdot \mathrm{Eval}(\tau \mid u) \tag{14}$$

As a result of the search, each $\mathcal{L}_{v,m,t}$ is partitioned into pairwise disjoint sets $\mathcal{L}_1, \ldots, \mathcal{L}_k$ in such a way that the sets $\mathcal{L}_i$ are in one-to-one correspondence with the items of $\mathcal{H}$. In Alg. 1, each $\mathcal{L}_i$ is represented by a tuple $(v,m,t,p,p_{\mathrm{grad}})$ where p and p_{grad} correspond to the value and the gradient of $\mathbf{P}^\sigma(\mathcal{L}_i)$ at σ, respectively. Then, writing $e = ((v',m'),(v,m))$, the value of $\mathbf{P}^\sigma(e,\tau)$ (cf. (12)) can be computed as the sum of $\mathbf{P}^\sigma(\mathcal{L}_{v,m,t})$ over all $t \le d(\tau) - time(v',v)$ where $\mathbf{P}^\sigma(\mathcal{L}_{v,m,t})$ is computed as the sum of $\mathbf{P}^\sigma(\mathcal{L}_i)$ over the sets $\mathcal{L}_i$ that form the partition of $\mathcal{L}_{v,m,t}$. The gradient is computed analogously. In Alg. 1, we also use two auxiliary arrays $\mathcal{V}$ and $\mathcal{G}$ for storing the value and the gradient of $\mathbf{P}^\sigma(\mathcal{L}_{v,m,t})$ for all $(v,m) \in \widehat{V}$ and the currently examined traversal time t.

The body of the main loop (lines 8–36) is executed for every $\tau \in T$ and computes $\mathbf{P}^\sigma(e,\tau)$ and $\nabla\mathbf{P}^\sigma(e,\tau)$ for every $e \in \widehat{E}$. The correctness of the algorithm follows from the fact that after executing line 14, for every $(v,m) \in \widehat{V}$ we have that

$$\mathbf{P}^\sigma(\mathcal{L}_{v,m,\ell}) = \sum_{(v,m,\ell,p,p_{\mathrm{grad}}) \in \mathcal{H}} p \tag{15}$$

and

$$\nabla\mathbf{P}^\sigma(\mathcal{L}_{v,m,\ell}) = \sum_{(v,m,\ell,p,p_{\mathrm{grad}}) \in \mathcal{H}} p_{\mathrm{grad}}, \tag{16}$$

which can be proved by induction on the number of iterations of the loop at lines 12–36 (the value of ℓ assigned at line 14 always increases between successive iterations).

Algorithm 1: Compute $\mathbf{P}^\sigma$ and $\nabla\mathbf{P}^\sigma$

input : A patrolling graph G, a regular strategy σ
output : The sets $\{\mathbf{P}^\sigma(e,\tau)\}$ and $\{\nabla\mathbf{P}^\sigma(e,\tau)\}$

```
1  P^σ  : array indexed by edges Ê and targets T
2  ∇P^σ : array indexed by edges Ê and targets T
3  V : array indexed by eligible pairs V̂
4  G : array indexed by eligible pairs V̂
5  H : min-heap of tuples (v, m, t, p, p_grad) sorted by t
6  set all elements of P^σ to 0 and ∇P^σ to 0⃗
7  foreach τ ∈ T do
8      set H to empty heap
9      foreach m such that (τ, m) ∈ V̂ do
10         H.insert(τ, m, 0, α(τ)β(τ), 0⃗)
11     end
12     while not H.empty do
13         set all elements of V to 0 and G to 0⃗
14         ℓ = H.peek.t
15         repeat
16             (v, m, t, p, p_grad) = H.pop
17             V(v, m) += p
18             G(v, m) += p_grad
19         until H.empty or H.peek.t > ℓ
20         foreach (v, m) such that V(v, m) > 0 do
21             foreach e = ((v', m'), (v, m)) ∈ Ê do
22                 t = time(v', v)
23                 if ℓ + t ≤ d(τ) then
24                     P^σ(e, τ) += V(v, m)
25                     ∇P^σ(e, τ) += G(v, m)
26                     p = V(v, m) · σ(e)
27                     if v' = τ then p *= 1 − β(τ)
28                     foreach e' ∈ Ê do
29                         p_grad(e') = σ(e) · G(v, m)(e')
30                         if e' = e then
                               p_grad(e') += V(v, m)
31                         if v' = τ then
                               p_grad(e') *= 1 − β(τ)
32                     end
33                     H.insert(v', m', ℓ + t, p, p_grad)
34                 end
35             end
36     end
37 end
38 return P^σ, ∇P^σ
```

Complexity analysis Let Λ be the total number of pairwise different t's for which there exists $u \in Path(e,\tau)$ with traversal time equal to t. Note that there are at most $|\widehat{E}| \cdot \Lambda$ items in $\mathcal{H}$, so each heap operation takes time $O(\log(|\widehat{E}| \cdot \Lambda))$. An analysis of the main loop (lines 8–36) reveals that the time complexity of Alg. 1 is

$$O\big(|T| \cdot |\widehat{E}|^2 \cdot \Lambda \cdot (|\widehat{E}| + \log(|\widehat{E}| \cdot \Lambda)\big)\,. \tag{17}$$

The size of Λ plays a crucial role. It stays reasonably small even if G contains "long" edges. In the worst case, Λ can be equal to $d_{\max}$, but this is rarely the case in practice.

Clearly, the traversal time of every $u \in Path(e, \tau)$ is at least $t_{\min} = \min_{v \to v'} time(v, v')$; and for many other t's between 1 and $d_{\max}$, there may exist no $u \in Path(e, \tau)$ with traversal time equal to t. This also explains why applying the algorithm of Klaška et al. [2018] to the modified graph obtained from G by splitting the long edges into sequences of unit-length edges is far less efficient than applying our algorithm directly to G. Such modification increases the number of vertices very quickly, even if Λ is small. This is confirmed experimentally in Sec. 5.

4.3 REGSTAR ALGORITHM

Having a fast, efficient and differentiable algorithm for strategy evaluation enables us to apply gradient ascent methods. For a given patrolling graph G and fixed memory sizes mem, we start with strategy σ having its values assigned randomly. Then, in an optimization loop, we examine $\mathrm{RVal}_G(\sigma)$ and modify σ in the direction of its gradient until no gain is achieved (see Alg. 2).

The Regstar algorithm constructs a Defender's strategy by running Alg. 2 repeatedly for a given number of random σ and selecting the best outcome. This results in high-quality strategies, as experimental results confirm.

Normalization By definition, a regular strategy σ is a bunch of probability distributions. A modification of σ by an update vector ξ can (and does) violate this property. A workable solution then requires the use of a normalization. Our normalization procedure $N_1(\sigma)$ crops all elements of σ into the interval $[0, 1]$ and then returns $\sigma(\widehat{v})/|\sigma(\widehat{v})|$ for every $\widehat{v} \in \widehat{V}$, where $|\sigma(\widehat{v})| = \sum_{(\widehat{v},v)\in\widehat{E}} \sigma(\widehat{v}, v)$. The function $\sigma \mapsto N_1(\sigma)$ is again differentiable at almost every point σ. Therefore, the composition of normalization and evaluation results in a differentiable algorithm whose gradient is, by the chain rule, the composition $\nabla \mathrm{RVal}_G(N_1(\sigma)) \cdot \nabla N_1(\sigma)$.

Prior works [Kučera and Lamser, 2016, Klaška et al., 2018] omitted this step assuming that the parameter space is the set of normalized strategies. They implicitly modify their gradients in order that the update results in a normalized strategy. This approach can be modelled in our setting by considering different "pivoted" normalization, denoted by $N_p(\sigma)$. It crops the values as well and, for $\widehat{v} \in \widehat{V}$, $N_p(\sigma)(\widehat{v}, v)$ equals to $\sigma(\widehat{v}, v)$ for all $(\widehat{v}, v) \in \widehat{E}$ except one "pivot", say v_p, for which $\sigma(\widehat{v}, v_p) = 1 - \sum_{(\widehat{v},v)\in\widehat{E}, v\neq v_p} \sigma(\widehat{v}, v)$.

Pivoting normalization N_p yields very sparse gradients ∇N_p in oppose to ∇N_1 and filters the signal propagation back to σ. This potentially slows down the optimization, as demonstrated in our experiments (see the supplementary material).

Minima softening So far, we supposed that $\mathrm{RVal}_G(\sigma)$ returns a value $\mathbf{P}^\sigma(e, \tau)$ that attains the minimal value. Therefore, only the top candidate is taken into account in the optimization. In contrast, one can consider more competitors $\mathbf{P}^\sigma(e, \tau)$ that are close to the minima and optimize for them simultaneously. We implement the very same "softening" method as in the baseline algorithm [Klaška et al., 2018].

Algorithm 2: Strategy optimization.

input : A patrolling graph G, a regular strategy σ
output : A regular strategy σ'

repeat
  $(\sigma, \nabla\sigma) = \texttt{Normalize}(\sigma)$
  $(p, \nabla p) = \mathrm{RVal}_G(\sigma)$
  $\sigma' = \texttt{Step}(\sigma, \nabla p \cdot \nabla\sigma)$
until $\mathrm{RVal}_G(\sigma') - p \leq threshold$
return σ'

Strategy update In each optimization step, we update the strategy σ by a proportion of the proper gradient $\xi = \nabla \mathrm{RVal}_G \cdot \nabla N_1$. We use the same scheduling as proposed by Klaška et al. [2018], where a variant of an update $\sigma' = \sigma + (1 - \delta)^k \xi$ is being used.

5 EXPERIMENTS

In many natural patrolling scenarios, the targets are distinguished geographic locations (banks, patrol stations, ATMs, tourist attractions, etc.). The connecting edges model the admissible moves of patrolling units (drones, police cars, etc.). Such graphs naturally contain edges of varying traversal time.

All the existing strategy improvement algorithms are designed for patrolling graphs with edges of unit traversal time. They can be applied to graphs with general topology once every "long" edge is replaced with a path consisting of edges of the unit length passing through fresh auxiliary vertices. We will apply this modification to graphs when necessary, without further notice.

In the first experiment, we compare the efficiency of Regstar against currently the best Baseline [Klaška et al., 2018]. In the second experiment, we demonstrate the capability of Regstar on a real-world patrolling problem that is far beyond the limits of Baseline. In the last experiment, we examine the impact of available memory size on the protection achieved by Regstar.

5.1 COMPARISON TO BASELINE

We consider patrolling graphs where ten targets are selected randomly from $n \times n$ grid. The traversal time between two

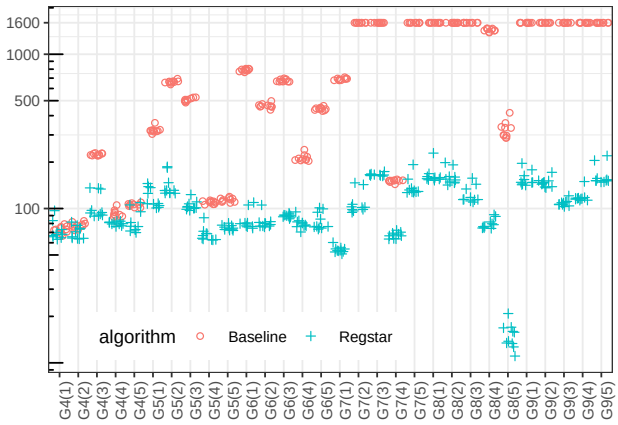

Figure 3: Comparison of REGSTAR against BASELINE. The time (in secs., logscale) needed to finish 50 optimization runs on various graphs is shown. The timeout is 1600 s.

vertices corresponds to their L^1 distance[5]. For each $n = 4, \ldots, 9$, we randomly select five graphs $G_n(1), \ldots, G_n(5)$. Thus, we obtain a collection of 30 patrolling graphs. Note that these graphs tend to contain longer edges with the increased n. The targets' cost is selected randomly between 180 and 200, and intrusion detection is perfect. The time needed to complete an attack is the same for all targets in $G_n(i)$, and it is set to a value for which the Defender can achieve a reasonably high protection ($d(\tau) = time_{\max} + time_{\text{avg}} + 3$, where $time_{\max}$ and $time_{\text{avg}}$ is the maximal and the average traversal time of an edge). For each $G_n(i)$, we report the total running time REGSTAR and BASELINE need to improve the same set of 50 randomly generated initial regular strategies. The timeout was set to 1600 s. This is repeated 10 times for every $G_n(i)$, outcoming 20 accumulated times shown in Fig. 3. Note that the time scale is *logarithmic*.

Observe that REGSTAR terminates in about 100–200 seconds in all cases while BASELINE reaches the timeout even for some $G_7(i)$ graphs and for *all* $G_9(i)$ graphs. This demonstrates that REGSTAR outperforms the BASELINE. When both algorithms terminate, the achieved protection values are about the same.

5.2 PATROLLING AN ATM NETWORK

We examine a patrolling graph where the vertices correspond to selected ATMs in Montreal (Fig. 1). All parameters of the graph are chosen in the same way as in Sec. 5.1, except for $d(\tau) = 2 * time_{\max} + time_{\text{avg}}$ and imperfect intrusion detection, which is set randomly to a value between 0.8 and 1.

[5]We use the L^1 (taxicab, Manhattan) distance instead of Euclidean distance because the former is considered as a better approximation of commuting distance.

Table 1: Analysis of REGSTAR on Montreal's ATMs.

m	$\text{RVal}_{\text{best}}$	RVal_{avg}	close (%)	iter	time (s)
1	64	57 ± 3	46	280 ± 30	5 ± 1
2	75	70 ± 2	83	684 ± 41	79 ± 8
3	80	77 ± 2	100	1045 ± 60	360 ± 58
4	81	79 ± 1	100	1346 ± 75	1250 ± 196

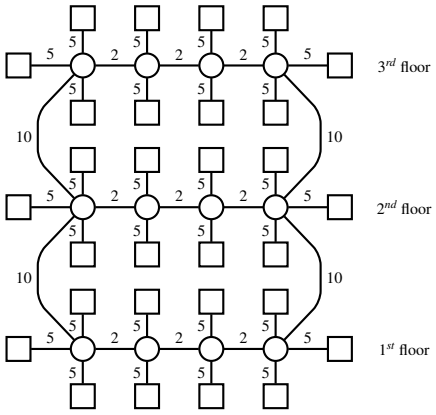

Figure 4: A building with three floors connected by stairs.

Note that the corresponding graph adjustment needed to run the BASELINE results in more than 3 thousand auxiliary vertices, far beyond its limits. Klaška et al. [2018] claim that BASELINE can solve instances with about 100 vertices.

The results achieved by REGSTAR are summarized in Tab. 1. For $m = 1, \ldots, 4$ we randomly generate 100 initial regular strategies where every vertex is assigned m memory elements. We report the best and the average protection value achieved by REGSTAR, the percentage of runs for which the resulting value reached at least 90% of the best value (labeled "close"), and the average number of iterations and time needed by one REGSTAR run.

Note that even for $m = 4$, the algorithm can improve one strategy in about 20 mins. The number of runs for which the optimization produces a high-value strategy is consistently very high and increases with m.

5.3 PATROLLING AN OFFICE BUILDING

We consider three office buildings with one, two, and three floors. On each floor, there are ten offices alongside a corridor. Stairs connect the floors on both sides of the corridors.

The building with three floors is shown in Fig. 4. The squares represent the offices, and the circles represent the corridor locations where the Defender may decide to visit the neighbouring offices. The "long" edges represent stairs. Every office's cost is set to 100, and the probability of successful intrusion detection is 0.9. The time needed to complete an intrusion is set to 100, 200, and 300 for the building with

Table 2: Analysis of REGSTAR on office buildings.

m	One floor			Two floors			Three floors		
	$RVal_{best}$	close (%)	time (s)	$RVal_{best}$	close (%)	time (s)	$RVal_{best}$	close (%)	time (s)
1	27	6	0.02 ± 0.01	32	29	0.6 ± 0.5	29	3	2 ± 1
2	41	28	1.2 ± 0.5	37	35	11 ± 8	32	5	10 ± 16
3	44	84	4.8 ± 2.1	45	30	47 ± 44	44	0.5	78 ± 142
4	47	83	11.7 ± 5.8	53	4	102 ± 118	44	3	238 ± 469
5	47	75	24.7 ± 14.2	56	12	236 ± 283	50	0.5	286 ± 635
6	47	65	45.2 ± 32.8	57	11	268 ± 420	55	0.5	641 ± 1557
7	51	45	74.8 ± 64.8	58	18	543 ± 877	53	1	1278 ± 3414
8	52	22	112.0 ± 111.8	59	9	515 ± 1116	56	1	1486 ± 4466

one, two, and three floors, respectively.

The outcomes achieved by REGSTAR are summarized in Tab. 2. For every building and every $m \in \{1, \ldots, 8\}$, 200 runs were processed with all nodes having m memory elements. We report the best value found, the percentage of runs for which the resulting value reached at least 90% of the best value (labeled "close"), and the average runtime.

We observe that the achieved protection substantially increases with more memory elements. This is because the considered patrolling graphs are relatively sparse, and "remembering" the history of previously visited vertices is inevitable for arranging optimal moves. In contrast, the patrolling graphs of ATM networks presented in Sec. 5.2 are fully connected, and the extra memory elements have not brought many advantages.

The REGSTAR algorithm discovers the relevant information about the history fully automatically. To examine this capability, we again consider the building with one floor, but we change the probability of successful intrusion detection to 1 and increase the attack time to 112 units. This is *precisely* the time the Defender needs to visit every target before an arbitrary attack is completed to achieve a *perfect* protection equal to 100. The Defender must schedule his walk to visit every office precisely once and with appropriate timing.

This experiment (see Tab. 3) show that REGSTAR can indeed discover this "clever" walk. For every $m \in \{1, \ldots, 8\}$, 500 runs were processed. The third column shows the percentage of runs resulting in a perfect protection strategy. Observe that *four* memory elements are sufficient to achieve perfect protection, and the chance of discovering a perfect strategy further increases with more memory elements.

Table 3: Reaching perfect protection for a one-floor office building with tight attack time.

m	$RVal_{best}$	close (%)	time (s)
1	34	0	0.05 ± 0.03
2	50	0	2.1 ± 1.0
3	55	0	8.6 ± 4.8
4	100	1.2	20 ± 13
5	100	5.0	37 ± 31
6	100	12.2	73 ± 67
7	100	11.8	107 ± 122
8	100	8.6	119 ± 180

6 CONCLUSION

Our efficient and differentiable regular-strategy evaluation algorithm proved to apply to patrolling graphs with arbitrary edge lengths and imperfect intrusion detection. The experimental results are encouraging and indicate that high-quality Defender's strategies can be constructed by optimization methods in a reasonable time for real-world scenarios.

Our experiments also show that the current optimization methods do not often converge to the best possible strategy. This suggests that an extra performance could be gained in exploration and improvement on the optimization side.

In Theorem 1, we proved that regular strategies *approximate* the optimal strategy to arbitrary precision $\varepsilon > 0$. The needed memory size depends on ε. We evaluated various memory sizes m experimentally from $m = 1$ up to the highest computable values in Sec. 5.2 and Sec. 5.3. Finding the best m (or even proving that regular strategies achieve the optimality) is a challenging open problem.

Acknowledgements

Research was sponsored by the Army Research Office and was accomplished under Grant Number W911NF-21-1-0189. This work was supported from Operational Programme Research, Development and Education - Project Postdoc2MUNI (No. CZ.02.2.69/0.0/0.0/18_053/0016952).

Disclaimer The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Office or the U.S. Government. The U.S. Government is authorized to reproduce and

distribute reprints for Government purposes notwithstanding any copyright notation herein.

References

N. Agmon, S. Kraus, and G. Kaminka. Multi-robot perimeter patrol in adversarial settings. In *Proceedings of ICRA 2008*, pages 2339–2345. IEEE Computer Society Press, 2008a.

N. Agmon, V. Sadov, G. A. Kaminka, and S. Kraus. The impact of adversarial knowledge on adversarial planning in perimeter patrol. In *Proceedings of AAMAS 2008*, pages 55–62, 2008b.

Noa Agmon, Sarit Kraus, Gal A Kaminka, and Vladimir Sadov. Adversarial uncertainty in multi-robot patrol. In *Proccedings of IJCAI 2009*, pages 1811–1817, 2009.

N. Basilico, N. Gatti, and F. Amigoni. Leader-follower strategies for robotic patrolling in environments with arbitrary topologies. In *Proceedings of AAMAS 2009*, pages 57–64, 2009.

N. Basilico, N. Gatti, and F. Amigoni. Patrolling security games: Definitions and algorithms for solving large instances with single patroller and single intruder. *Artificial Inteligence*, 184–185:78–123, 2012.

A. Beynier. A multiagent planning approach for cooperative patrolling with non-stationary adversaries. *International Journal on Artificial Intelligence Tools*, 26(5), 2017.

B. Bošanský, V. Lisý, M. Jakob, and M. Pěchouček. Computing Time-Dependent Policies for Patrolling Games with Mobile Targets. In *Proceedings of AAMAS 2011*, pages 989–996, 2011.

B. Bošanský, O. Vaněk, and M. Pěchouček. Strategy Representation Analysis for Patrolling Games. In *Proceedings of AAAI Spring Symposium 2012*, pages 9–12, 2012.

T. Brázdil, P. Hliněný, A. Kučera, V. Řehák, and M. Abaffy. Strategy synthesis in adversarial patrolling games. *CoRR*, abs/1507.03407, 2015.

T. Brázdil, A. Kučera, and V. Řehák. Solving patrolling problems in the internet environment. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI 2018)*, pages 121–127, 2018.

E. Munoz de Cote, R. Stranders, N. Basilico, N. Gatti, and N. Jennings. Introducing alarms in adversarial patrolling games: extended abstract. In *Proceedings of AAMAS 2013*, pages 1275–1276, 2013. ISBN 978-1-4503-1993-5.

J. Gan, B. An, Y. Vorobeychik, and B. Gauch. Security games on a plane. In *Proceedings of AAAI 2017*, pages 530–536. AAAI Press, 2017.

J. Gan, E. Elkind, and M. Wooldridge. Stackelberg security games with multiple uncoordinated defenders. In *Proceedings of AAMAS 2018*, pages 703–711, 2018.

M. Jain, E. Karde, C. Kiekintveld, F. Ordóñez, and M. Tambe. Optimal defender allocation for massive security games: A branch and price approach. In *Workshop on Optimization in Multi-Agent Systems at AAMAS*, 2010.

J. Karwowski, J. Mandziuk, A. Zychowski, F. Grajek, and B. An. A memetic approach for sequential security games on a plane with moving targets. In *Proceedings of AAAI 2019*, pages 970–977, 2019.

C. Kiekintveld, M. Jain, J. Tsai, J. Pita, F. Ordóñez, and M. Tambe. Computing optimal randomized resource allocations for massive security games. In *Proceedings of AAMAS 2009*, pages 689–696, 2009.

D. Klaška, A. Kučera, T. Lamser, and V. Řehák. Automatic synthesis of efficient regular strategies in adversarial patrolling games. In *Proceedings of AAMAS 2018*, pages 659–666, 2018.

D. Klaška, A. Kučera, and V. Řehák. Adversarial patrolling with drones. In *Proceedings of AAMAS 2020*, pages 629–637, 2020.

D. Klaška, A. Kučera, T. Lamser, and V. Řehák. Automatic synthesis of efficient regular strategies in adversarial patrolling games. In *Proceedings of AAMAS 2018*, pages 659–666, 2018.

A. Kučera and T. Lamser. Regular strategies and strategy improvement: Efficient tools for solving large patrolling problems. In *Proceedings of AAMAS 2016*, pages 1171–1179, 2016.

E. S. Lin, N. Agmon, and S. Kraus. Multi-robot adversarial patrolling: Handling sequential attacks. *Artificial Intelligence*, 274:1 – 25, 2019. ISSN 0004-3702.

J. Pita, M. Jain, J. Marecki, F. Ordónez, C. Portway, M. Tambe, C. Western, P. Paruchuri, and S. Kraus. Deployed ARMOR protection: The application of a game theoretic model for security at the Los Angeles Int. Airport. In *Proceedings of AAMAS 2008*, pages 125–132, 2008.

A. Sinha, F. Fang, B. An, C. Kiekintveld, and M. Tambe. Stackelberg security games: Looking beyond a decade of success. In *Proceedings of IJCAI 2018*, pages 5494–5501. ijcai.org, 2018.

M. Tambe. *Security and Game Theory. Algorithms, Deployed Systems, Lessons Learned*. Cambridge University Press, 2011.

J. Tsai, S. Rathi, C. Kiekintveld, F. Ordóñez, and M. Tambe. IRIS—a tool for strategic security allocation in transportation networks categories and subject descriptors. In *Proceedings of AAMAS 2009*, pages 37–44, 2009.

Y. Vorobeychik, B. An, and M. Tambe. Adversarial patrolling games. In *Proceedings of AAAI 2012*, pages 91–98, 2012.

Y. Wang, Z.R. Shi, L. Yu, Y. Wu, R. Singh, L. Joppa, and F. Fang. Deep reinforcement learning for green security games with real-time information. In *Proceedings of AAAI 2019*, pages 1401–1408, 2019.

H. Xu, A. X. Jiang, A. Sinha, Z. Rabinovich, S. Dughmi, and M. Tambe. Security games with information leakage: Modeling and computation. In *Proceedings of IJCAI 2015*, pages 674–680, 2015.

H. Xu, K. Wang, P. Vayanos, and M. Tambe. Strategic coordination of human patrollers and mobile sensors with signaling for security games. In *Proceedings of AAAI 2018*, pages 1290–1297, 2018.

Z. Yin, D. Korzhyk, C. Kiekintveld, V. Conitzer, and M. Tambe. Stackelberg vs. Nash in security games: Interchangeability, equivalence, and uniqueness. In *AAMAS*, pages 1139–1146, 2010.